REMARKS

In the final Office Action, the Examiner rejected claims 1, 3-5, 7-9, 11, 13, 15, 16, 18-22, and 24-35 under 35 U.S.C. § 112, first paragraph, as failing to comply with the enablement requirement; and rejected claims 1, 3-5, 7-9, 11, 13, 15, 16, 18-22, and 24-35 under 35 U.S.C. § 102(b) as being clearly anticipated by the documentation of REOHR (Copy of deposited work registered in the U.S. Copyright Office under number "Txu 790-574," entitled "A VIRTUALIZED NETWORK", Jan. 3, 1997).

By this Amendment, Applicant proposes canceling claims 1, 3, 5, 7-9, 11, 13, 15, 16, 18-22, and 24-28, without prejudice or disclaimer of the subject matter thereof, and amending claims 31 and 35 to improve form. Applicant previously canceled claims 2, 4, 6, 10, 12, 14, 17, and 23. Applicant respectfully traverses the Examiner's rejections under 35 U.S.C. §§ 102(b) and 112. Claims 29-35 would be pending upon entry of the present amendment.

REJECTION UNDER SECTION 112, FIRST PARAGRAPH

In paragraph 4 of the Office Action, the Examiner rejected claims 1, 3-5, 7-9, 11, 13, 15, 16, 18-22, and 24-35 under 35 U.S.C. § 112, first paragraph, as failing to comply with the enablement requirement. Applicant respectfully traverses the Section 112, first paragraph, rejection with regard to the claims presented herein.

With regard to the Section 112, first paragraph, rejection, the Examiner alleged in paragraph 5 of the Office Action:

> Applicant specification set forth high level examples of the desired end results and label for which the elements the claims are to be referenced as, applicant has left it to other to supply the under lying glue by which the System requires to operate. Therefore applicant has failed to supply a disclosure by which one of ordinary skill in the art can make or use the invention.
>
> Example claim 1, "permitting direct automation of real world functions without the prior

> systemization of the real world functions," this passage is enabled in the specification with numerous paraphrasing of the statement.

Applicant respectfully disagrees and submits that one skilled in the art could make or use Applicant's invention, as recited in claims 29-35, in light of the Applicant's disclosure without undue experimentation.

The enablement requirement refers to the requirement of 35 U.S.C. § 112, first paragraph that the specification describe how to make and how to use the invention. The invention that one skilled in the art must be enabled to make and use is that defined by the claim(s) of the particular application or patent. However, to comply with 35 U.S.C. § 112, first paragraph, it is not necessary to "enable one of ordinary skill in the art to make and use a perfected, commercially viable embodiment absent a claim limitation to that effect." *CFMT, Inc. v. Yieldup Int'l Corp.*, 349 F.3d 1333, 1338, 68 U.S.P.Q.2d 1940, 1944 (Fed. Cir. 2003). Detailed procedures for making and using the invention may not be necessary if the description of the invention itself is sufficient to permit those skilled in the art to make and use the invention. M.P.E.P. § 2164.

The standard for determining whether the specification meets the enablement requirement was cast in the Supreme Court decision of *Mineral Separation v. Hyde*, 242 U.S. 261, 270 (1916) which postured the question: is the experimentation needed to practice the invention undue or unreasonable? That standard is still the one to be applied. *In re Wands*, 858 F.2d 731, 737, 8 U.S.P.Q.2d 1400, 1404 (Fed. Cir. 1988).

In making a determination that a disclosure does not satisfy the enablement requirement and whether any necessary experimentation is "undue," there are several factors that must be considered (see M.P.E.P. §2164.01(a)). These factors include the breadth of the claims, the nature of the invention, the state of the prior art, the level of one of ordinary skill, the level of

predictability in the art, the amount of direction provided by the inventor, the existence of working examples, and the quality of experimentation needed to make or use the invention based on the content of the disclosure. M.P.E.P. § 2164.01(a). As required by *In re Wands*, 858 F.2d at 740, 8 USPQ2d at 1407, the Examiner must consider all the evidence related to each of these factors, and any conclusion of nonenablement must be based on the evidence as a whole.

Applicant respectfully submits that the Examiner has not considered any of the factors required by *In re Wands* and, as such, the Examiner's rejection of claims 29-35 under 35 U.S.C. § 112, first paragraph, based on enablement is improper. Applicant further submits that the Section 112, first paragraph, rejection is improper because the Examiner did not individually address each independent claim alleged to be non-enabled. Rather, the Examiner only cited a portion of independent claim 1. Nonetheless, Applicant submits that claims 29-35 are enabled by Applicant's disclosure because several *In re Wands* factors are satisfied.

A. <u>The Breadth Of The Claims</u>

The breadth of the claims is one factor to consider pursuant to *In re Wands*. This requires that the Examiner determine exactly what subject matter is encompassed by the claims. *AK Steel Corp. v. Sollac*, 344 F.3d 1234, 1244, 68 U.S.P.Q.2d 1280, 1287 (Fed. Cir. 2003). The Federal Circuit has repeatedly held that "the specification must teach those skilled in the art how to make and use the full scope of the claimed invention without 'undue expermimentation'." The determination of the propriety of a rejection based upon the scope of a claim relative to the scope of the enablement involves two stages of inquiry. The first is to determine how broad the claim is with respect to the disclosure. The second inquiry is to determine if one skilled in the art is enabled to make and use the entire scope of the claimed invention. M.P.E.P. § 2164.08.

Applicant submits that the scope and breadth of claims 29-35 satisfies the enablement requirement of Section 112, first paragraph. For example, independent claim 29 is directed to a system for providing real to virtual correspondence so that premeditated and definable functions performed by a real world entity may be mimicked by a counterpart entity program in a virtual world of machine memory. The system includes a memory configured to store virtualized network (VN) adaptation logic, including a virtualized entity (VENT) table, the VN adaptation logic providing a software environment in which a plurality of entity programs are executed, each entity program matching one-to-one to a counterpart, real world entity. The system also includes a processor configured through the VN adaptation logic to execute instructions to cause the execution of any entity program in the software environment whenever the entity program receives data and action from another entity program, and respond to a speak request of a currently executing entity program by passing data and action from the speaking entity program to a listening entity program addressed by the speaking entity program, enabling virtual-to-virtual interactions.

The specification properly sets forth how to make or use the elements of the system of claim 29 (and dependent claims 30-35), without undue experimentation.

1. Memory And Processor

Claim 29 recites that the system includes a memory and a processor configured to execute instructions stored in the memory. Pages 17-19 of the specification provide:

> Other entities may be virtualized into the network. A device-type VENT may include software, such as programs, threads, processes, information, databases, or objects; hardware, such as a computer, a laptop, a personal digital assistant (PDA), a wired or wireless telephone, or a similar wireless device; or a combination of both software and hardware.
>
> * * *

> As used herein the term "stored program machine" includes any machine such as a conventional computing machine (e.g., a computer) that includes a bus interconnecting a processor, and a main memory. As shown in Fig. 2, an entity, whether it be a client entity 104, a server entity 106, or a client/server entity 108, includes a bus 200 interconnecting a processor 202, a read-only memory (ROM) 204, a main memory 206, a storage device 208, an input device 210, an output device 212, and a communication interface 214. Bus 200 is a network topology or circuit arrangement in which all devices are attached to a line directly and all signals pass through each of the devices. Each device has a unique identity and can recognize those signals intended for it. Processor 202 includes the logic circuitry that responds to and processes the basic instructions that drive entity 104, 106, 108. ROM 204 includes a static memory that stores instructions and date used by processor 202.
>
> * * *
>
> As will be described below, an entity 104, 106, 108 consistent with the present invention may allow the functions of an entire organization to be gradually automated as each employee, independently, automates his/her own functions. Entity 104, 106, 108 performs this task in response to processor 202 executing sequences of instructions contained in a computer-readable medium, such as main memory 206. A computer-readable medium may include one or more memory devices and/or carrier waves.
>
> Execution of the sequences of instructions contained in main memory 206 causes processor 202 to perform processes that will be described later. Alternatively, hardwired circuitry may be used in place of or in combination with software instructions to implement processes consistent with the present invention. Thus, the present invention is not limited to any specific combination of hardware circuitry and software.
>
> * * *
>
> A mirror illustrates certain aspects of the entity correspondence of the present invention. Let the images, reflected by the mirror, correspond to the program entities within the stored program environment. Then, for each physical entity in the view of the mirror there is exactly one reflected image, or a simple 1-to-1 match. In total, the virtualized network reflects the entire desired portion of the real world into the stored program machine. In any event, for every real entity, there is a counterpart program entity (virtualized entity) defined into the stored program machine, which will then able to act for its counterpart.
>
> Entity correspondence is straightforward. For each real entity, a counterpart is virtualized or defined. When a virtualized network spreads beyond enterprise, beyond community with virtualized network nodes all over the globe, there will be only one program corresponding to the virtualized "me" entity. Absolute 1-for-1 matching is the ideal correspondence between real and virtual world counterparts.

These pages of the specification more than sufficiently specify the exemplary types of hardware (e.g., memory and processor) that may be used to implement the system recited in claims 29-35. For example, the system recited in these claims may include any entity 104, 106,

or 108, and "entity 104, 106, 108 consistent with the present invention may allow the functions of an entire organization to be gradually automated as each employee, independently, automates his/her own functions. Entity 104, 106, 108 performs this task in response to processor 202 executing sequences of instructions contained in a computer-readable medium, such as main memory 206."

2. Virtualized Network (VN) Adaptation Logic

Claim 29 recites that the memory is configured to store virtualized network (VN) adaptation logic. Several portions of the specification discuss VN adaptation logic. For example, page 9, line 15 – page 10, line 1 of the specification states:

> A software kernel is a program itself and is the logical extension or adaptation of a programmable machine. A kernel adapts a computer to provide a particular systems environment, within machine memory, for operating system, middleware, and application system programs. Technically designed and developed systems are the essential characteristic of computers.
>
> The different adaptation of this invention is a program also like the software kernel. However, within machine memory, it provides an environment for virtualized entities rather than an environment for systems. Whereas a computer kernel directs machine execution among the various technically designed systems contained therein, this different adaptation directs machine execution among the various virtualized entities contained therein.

This section of the specification states that the VN adaptation logic is similar to a software kernel, except that VN adaptation logic provides an environment for virtualized entities (not systems) within memory.

Page 22, line 6 – page 25, line 21 of the specification states:

> … an adaptation to the stored program machine -- a logical construct enabling necessary entity types, behaviors, actions, situations, roles, and relationships to be faithfully mapped from the virtualized world to the stored program machine architecture.
>
> Starting with the VN adaptation of the stored program machine, the present disclosure will describe its workings for virtual world interactions -- the program entity-to-program entity interactions. Thereafter, the present disclosure will describe its workings for

virtual world-to-physical world interactions -- intra-actions between the program entity and its physical counterpart.

* * *

Beginning with the VN adaptation and by way of example, imagine it is August of the year 2001. Hank and Beth, friends in a small town, decide to join an investment club. The club requires that members be virtualized into the virtualized network of the present invention. Beth manages all Teller Operations for a local bank. The bank utilizes VN. As a bank employee she has access to market information, and recently Hank has called several times daily. He always asks, "what's up"-- and means by that question --- what are the latest DOW and NASDAQ stock averages?

Beth already is virtualized into VN. This allows her to sponsor and actually enter Hank's virtualizing data into VN. Within hours, after she virtualizes Hank, he discovers he can use a terminal to pose his "what's up" question to Beth.

Fig. 3-1 looks inside VN to see how this works, remembering the virtual environment is created by the adaptation to the stored program machine. Memory space within the stored program machine is illustrated in Fig. 3-1. The entry/virtualization of Hank causes his record to be entered into the VENT table, one of four tables used by VN. It contains a pointer to his address in stored program machine memory (virtual space). All program information associated with Hank, now or in the future, will go to this place in virtual space. The same is true for Beth except she was earlier virtualized – so she already had her VENT table entry with its pointer to her virtual space. That is why she was able to sponsor Hank and enter his information.

1. Virtual to Virtual Interactions

When Hank's VENT is executing on his behalf, and it interacts with (speaks to) Beth's VENT, then VN logic, what was termed the stored program machine adaptation, effects the "speak" interaction since it is the environment of the virtual world. It looks-up Beth in the VENT table, finds her memory address in virtual space and moves the resultant spoken data to Beth's memory region. It starts her VENT executing with a pointer to the data packet (PAKT), the second of four table types used by VN. While the VENT table is global, one table for each node, there are many PAKT tables per node. In her region the PAKT looks something like:

BETH | What's up? | HANK

This is the very simple and very straightforward manner in which any virtualized entity "speaks" to any other. "Speak-listen" is the way virtualized entities interact.

Because of Hank's earlier behavior, Beth had pre-meditated this particular occurrence. When her virtualized self or VENT starts executing it knows what to do. It sees "What's up", and sees Hank was the speaker. Its action at this point is premeditated. There is no reason to reference the conscious Beth, so it acts for her, below her conscious level, speaking back to Hank as follows:

HANK | Dow=10354;NASDAQ=2371 | BETH

VN already knows Hank's region in virtual space as it was contained in the PAKT from him. So, without even looking him up in the VENT table, VN moves this data to Hank's memory region. It causes his VENT to begin execution at the right place, and as before,

provides a pointer to this answering PAKT.

Summarizing virtual to virtual interaction: A virtualized entity (VENT) executes as programs always have executed. During execution a VENT takes action, with regard to anything outside itself, by "speaking" as we have just seen. VN accepts the data contents from the speaking VENT. It always knows the "speaker", and knows the "listener" (unless it is the first "speak" of a conversation when it must look-up the listener's location in a VENT table). VN moves the data to the "listener's" region, and, when it has no higher priority, passes execution to that listener VENT. In this fashion, VN passes data and action from one VENT, whether person or device, to another VENT, either person or device. Every stored program machine has an identical adaptation and everyone operates as above, repetitively, and at all times, for all interactions by all VENTs within its virtual space.

When a VENT analyzes, i.e., as it identifies a situation and prepares to act, it does so quite similarly to the programs that have been used over the years. The only change here is that VN uses/supports decision table type "thinking", first identifying a situation as it was pre-meditated, then taking action(s) according to the premeditated order of those actions. This is much closer to the way people actually think. It is just the way one would write instructions for another person, telling that person how to identify and act in a particular situation -- what to look for to identify the situation, and then upon finding it, telling it precisely what to do.

However, when the analysis is complete, and the VENT acts, when it steps out of itself to interact with other entities, then its actions exactly mimic its real world counterpart. A VENT never speaks to a "system," to a "processor/computer," to "information or data," or any artificial systems or technical entity. A VENT always speaks to another VENT. Of course, each matches 1-for-1 with its counterpart.

For example, when Beth virtualized Hank by providing certain data, she/her VENT never spoke to the stored program machine (processor or computer in today's terminology). Recall a stored program machine, in combination with the adaptation, is not an entity; it is an environment. No, her VENT spoke to another VENT. Specifically, it spoke to the person, the VENT of the person, who manages her bank's own stored program machine. That person's VENT most likely included premeditated handling for the virtualization of a new entity because that is a normal responsibility of the VN machine manager. The machine manager's virtualized self already "knew" that bank policy permitted virtualization of a non-employee, on the bank's stored program machine, if no other machine was specified. As Hank's virtualization was normal the machine manager's VENT acted without his conscious involvement. That part of the stored program machine manager's responsibility was automated.

These pages of the specification not only provide a working example (discussed below), but also provide a detailed description of how the VN adaptation logic may be implemented in a virtual-to-virtual interaction. For example, the above pages disclose that the VN adaptation logic

accepts data contents from a speaking entity program (e.g., a VENT), moves the data to the listener VENT, and, when it has no higher priority, passes execution to that listener VENT. In this fashion, the VN adaptation logic passes data and action from one VENT to another VENT. Furthermore, the VN adaptation logic moves the data between the speaker and listener VENTs based on entries in the VENT table and pointers to VENT table entries.

Pages 26-44 of the specification provide further working examples of the VN adaptation logic, as well as descriptions of:

(1) how the VN adaptation logic may enable the entity programs (e.g., the VENTs) to interact by using bit strings, binary logic, and electrical transmission (page 27, lines 18-19 and page 28, lines 15-16) and by encoding the decision (e.g., VENT) tables (page 30, lines 12-13);

(2) how the VN adaptation logic may provide an entity program the needed connection each time it speaks to another entity program by disclosing that the VN adaptation logic transmits data in the form of a bit string, and moves the data into the space of a listener entity program, which may act on the data (page 30, line 23 – page 31, line 4);

(3) how the VN adaptation logic may be implemented in a virtual-to-physical interaction, e.g., by creating a channel between communications or physical devices, by moving the PAKT, or data and action, through the channel from virtual to physical entities, or vice versa (page 31, line 18 – page 37, line 4);

(4) how the VN adaptation logic may provide numerous channel capabilities (page 37, line 5 – page 43, line 22); and

(5) a summary of the VN adaptation logic (page 43, line 23 – page 44, line16).

Pages 44-48 of the specification provide a detailed description of the program structure

of the VN adaptation logic, and states:

> Because speak-listen is a repetitive action, occurring millions upon millions of times, the supporting logic of the adaptation must be simple and spare. Fig. 3-9 shows adaptation logic modules in dark lines while the VENT modules are shown in light lines. Modules above the dotted line are specified and implemented as decision tables (DTBLs), while those below are designed and encoded conventionally.
>
> When a VENT wishes to act or interact, it speaks as shown above and execution is assumed or taken by the TAKE module. TAKE validates the PAKT, and if this is the first speak of a conversation, it looks the listener up in the VENT table, finding its virtual space coordinates. The GIVE module receives execution, and in turn, passes that execution to the correct "listener" VENT, thus supporting a VENT's wish to speak which is equivalent to supporting all VENT actions necessary to mimic its physical counterpart.
>
> As described above, the passing of execution or action is apparently simple, but the passing of data is more complicated. While one entity may choose to speak, the other may not choose to listen. Just as in the physical world one may hear a noise from another but allot no mental (brain) space for what that other is saying. The same is true for our virtualized selves. The listener must have virtual space allocated for the specific speaker's PAKT whether it is one word or, at the speaker's choice, the entire encyclopedia. Below describes how a virtualized person keeps different topics, and the individual conversations within those topics, separate and distinct. Orthogonal VN ensures this condition exists before passing action to the listener.
>
> Space allocation, or memory management, has been the most difficult and crucial problem for operating system software. The greater the number of active tasks, the greater the number of needed program modules, and the more difficult the memory management/storage allocation problem is.
>
> Historically, for an operating system of any generality, one might observe that this problem was not solved, but compromised. Program size was controlled. A unit of work was accomplished by running several smaller programs one after another. Alternatively, a limit was placed upon the number of simultaneous tasks, since each task required its own resident data structure as well as the task program.
>
> Without controls on program size and limits on the number of simultaneous tasks, designers believed the stored program machine would be burdened. Excessive memory swapping, a condition called thrashing, was the problem to be avoided. So program size and the management of memory, by an operating system, was balanced against the allowed number of tasks and the management of those tasks. Larger programs meant fewer simultaneous tasks and vice versa. Memory management was viewed as interdependent with task management, snarling the logic of one with the logic of the other. Generally, the computer culture assumed such interdependence. Hence the need to compromise was an obvious, self-evident truth.
>
> A VENT program cannot be infinite in size, nor can an infinite number of tasks (with their associated data structures) be accommodated because there is not an infinite amount of disk (real) storage in the world. But, VN is designed without arbitrary, up-front limits.

To the contrary, every aspect is designed to be independent. Consequently, tasks and program size are strictly orthogonal. For example, a single virtualized node may contain many VENTs, and will likely have many tasks running at every point in time because each speak creates a separate task for that node. The number of potentially simultaneous tasks is high. However, a node will reach its operating limit much sooner since some percentage of interactions involve VENTs on other nodes. A reasonable limit is some multiple of the useable channels or pathways to/from the node. A program entity's space will be approximately 64 terabytes.

VN creates its own virtual space with its own coordinate system. For example, a VENT in the VENT TABLE contains a "pointer" to its region in virtual space. The "pointer" is given in virtual space coordinates. VN maps this virtual space back, via logic modules called PRIMatives (see Fig. 3-9), to the stored program machine memory as presented through the hardware memory-mapping architecture.

If too much virtual space is utilized or written, the stored program machine manager's VENT will be notified that disk limits are approaching. This is a problem of virtual space size only. On the other hand, if too many simultaneous "speaks" occur, or too much space swapping is required, the speed of the machine could slow.

These two situations were contemplated, and viewed as independent from one another. But neither was "solved" or compromised during the design of the adaptation. They are properly and easily solved later, when and if they occur. If space limits are approached, disk capacity is added. If throughput or response time degrades, the hardware platform is replaced with a faster, larger one, or the work is split among several different virtualized nodes. VN makes configuration changes such as these very easily.

If an Intel 386 machine is used as a platform, then there will be only 64 terabytes of programmer memory with hardware assists to map it back to 4 gigabytes real. VN provides just enough virtual space for devices, but for person entities it will provide approximately 64 terabytes. Thus, premeditated behavior of a person may grow greatly over time, independent of any other change in the physical or virtual worlds.

Large space makes additional premeditation very simple. And it is easy for the programmer to encode. He/she encodes only in-program instructions, but no program or I/O calls. He/she encodes only "speaks" and passively assents to "listens."

Returning to Fig. 3-9, note that the VNUC module is not a VENT. But, it is written as a VENT and has the identical speak-listen powers of a VENT. VNUC handles virtual memory management, at the logical level, ensuring the appropriate regions of virtual space are present in machine memory at appropriate times.

Memory management must transit hardware addressing logic, memory boundaries, and then secondary storage, before all is prepared for PAKT movement and the passing of execution to the listener VENT. This requires the issuance of machine-privileged instructions accomplished by the four short PRIMitive routines shown in Fig. 3-9. They exploit the machine to the fullest and make it conform/submit to the needs of the VN environment. In moving VN to a new type of hardware platform, only these short routines need to be re-written.

> The DRVR module of Fig. 3-9 is the decision table driver, the run-time piece of code that passes execution. Execution is first passed to the series of tests in a decision table (see Fig. 3-10). DRVR saves the yes/no test outcome from each and then determines the appropriate "rule" from all the outcomes. After that DRVR directs execution to each action in sequence as indicated by the rule.
>
> All VN shown above the dotted line of Fig. 3-9 is specified via decision tables and will be encoded as such. When execution is passed to the TAKE module, for instance, actual execution is passed to the DRVR with a pointer to the TAKE DTBL. The DRVR ensures all correct tests and actions are accomplished.
>
> The resultant independent pieces of logic mean that one may be changed without, unwittingly, changing any other. Improvements or fixes are easy.
>
> The data structures used by the adaptation are simple tables. Four tables are maintained at the logical level, two of them have already been discussed. A simple virtual to (programmer's) memory table, in combination with the usual hardware memory mapping tables, are also used.

These pages of the specification provide a detailed description of how the VN adaptation logic may include a TAKE module that validates a PAKT (e.g., data), and, if this is the first speak of a conversation, looks the listener up in the VENT table, finding its virtual space coordinates. The VN adaptation logic may also include a GIVE module that receives the PAKT, and passes the PAKT to the correct "listener" VENT. The VN adaptation logic may create virtual space with its own coordinate system, by mapping this virtual space back, via logic modules called PRIMatives (see Fig. 3-9), to the stored program machine memory as presented through the hardware memory-mapping architecture.

As further described in these pages of the specification, the VN adaptation logic may include a DRVR module, which is the decision table driver or the run-time piece of code that passes execution. When execution is passed to the TAKE module, for instance, actual execution is passed to the DRVR module with a pointer to the TAKE DTBL. The DRVR module may ensure that correct tests and actions are accomplished.

3. Virtualized Entity (VENT) Table

Claim 29 recites that the memory is configured to store a virtualized entity (VENT) table. Several portions of the specification discuss the VENT table. For example, page 23, lines 6-13 of the specification describes Fig. 3-1 and states:

> Fig. 3-1 looks inside VN to see how this works, remembering the virtual environment is created by the adaptation to the stored program machine. Memory space within the stored program machine is illustrated in Fig. 3-1. The entry/virtualization of Hank causes his record to be entered into the VENT table, one of four tables used by VN. It contains a pointer to his address in stored program machine memory (virtual space). All program information associated with Hank, now or in the future, will go to this place in virtual space. The same is true for Beth except she was earlier virtualized – so she already had her VENT table entry with its pointer to her virtual space. That is why she was able to sponsor Hank and enter his information.

This page of the specification and Fig. 3-1 provides a detailed description of what a VENT table may include. For example, as shown in Fig. 3-1, the VENT table may include a record for each VENT (e.g., entity program) name known by the stored program machine. The record for each VENT name may contain a pointer to its location in virtual space as well as other entity information.

4. Cause The Execution Of Any Entity Program

Claim 29 recites that the processor causes the execution of any entity program in the software environment whenever the entity program receives data and action from another entity program. Several portions of the specification enable this feature of claim 29. For example, page 23, lines 15-20 of the specification states:

> When Hank's VENT is executing on his behalf, and it interacts with (speaks to) Beth's VENT, then VN logic, what was termed the stored program machine adaptation, effects the "speak" interaction since it is the environment of the virtual world. It looks-up Beth in the VENT table, finds her memory address in virtual space and moves the resultant spoken data to Beth's memory region. It starts her VENT executing with a pointer to the data packet (PAKT), the second of four table types used by VN.

This portion of the specification discloses that when data (e.g., PAKT) is sent from one entity

program (e.g., Hank's VENT) to another entity program (e.g., Beth's VENT), the VN adaptation logic starts the execution of Beth's VENT with a pointer to the data.

5. Respond To A Speak Request

Claim 29 recites that the processor responds to a speak request of a currently executing entity program by passing data and action from the speaking entity program to a listening entity program addressed by the speaking entity program, enabling virtual-to-virtual interactions. Several portions of the specification enable this feature of claim 29. For example, page 24, lines 12-21 of the specification states:

> Summarizing virtual to virtual interaction: A virtualized entity (VENT) executes as programs always have executed. During execution a VENT takes action, with regard to anything outside itself, by "speaking" as we have just seen. VN accepts the data contents from the speaking VENT. It always knows the "speaker", and knows the "listener" (unless it is the first "speak" of a conversation when it must look-up the listener's location in a VENT table). VN moves the data to the "listener's" region, and, when it has no higher priority, passes execution to that listener VENT. In this fashion, VN passes data and action from one VENT, whether person or device, to another VENT, either person or device. Every stored program machine has an identical adaptation and everyone operates as above, repetitively, and at all times, for all interactions by all VENTs within its virtual space.

This portion of the specification discloses that the VN adaptation logic accepts the data and action from the speaking entity program (e.g., speaking VENT), and moves the spoken data and action to the listening entity program (e.g., the listener VENT) to provide virtual-to-virtual interactions.

In light of the above, Applicant submits that the specification adequately sets forth how to make or use the features of the system of claim 29, without undue experimentation. Claims 30-35 depend from claim 29, and are enabled for at the least the reasons set forth above for claim 29. Applicant respectfully submits that this *In re Wands* factor (e.g., the breadth of the claims) supports the enablement of claims 29-35.

B. The Inventor Provided Ample Direction

The amount of guidance or direction needed to enable an invention is inversely related to the amount of knowledge in the state of the art as well as the predictability in the art. *In re Fisher*, 427 F.2d 833, 839, 166 U.S.P.Q. 18, 24 (C.C.P.A. 1970). The "amount of guidance or direction" refers to that information in the application that teaches exactly how to make or use the invention. The more that is known in the prior art about the nature of the invention, how to make, and how to use the invention, and the more predictable the art is, the less information needs to be explicitly stated in the specification. M.P.E.P. § 2164.03.

The specification is replete with examples and descriptions of how to make and use the system recited in claims 29-35, as shown above. Thus, Applicant submits that the specification provides a significant amount of guidance and direction to enable one to make and use the system recited in claims 29-35. Applicant provided such detailed guidance despite the fact that the communications and computer art is predictable, and knowledge in the art is great. Thus, Applicant respectfully submits that these *In re Wands* factors (e.g., the level of one of ordinary skill, the level of predictability in the art, and the amount of direction provided by the inventor) support the enablement of claims 29-35.

C. Working Examples

The existence of working examples is another *In re Wands* factor that weighs in favor of enablement, although the specification need not contain an example. *In re Borkowski*, 422 F.2d 904, 908, 164 U.S.P.Q. 642, 645 (C.C.P.A. 1970). As shown in the above-cited sections of the specification, Applicant has provided numerous examples showing how to make and use the system recited in claims 29-35. For instance, examples may be found at page 22, line 15 – page

25, line 21; page 25, line 22 – page 27, line 6; page 31, line 22 – page 32, line 11; page 33, line 5 – page 36, line 20; page 37, line 16 – page 39, line 1; page 41, lines 6-19; and page 48, line 20 – page 58, line 21.

Applicant respectfully submits that this *In re Wands* factor (e.g., the existence of working examples) further supports the enablement of claims 29-35.

In light of above, Applicant submits that the Examiner has not considered the factors required by *In re Wands* with respect each of the pending claims and, as such, the Examiner's rejection of these claims under 35 U.S.C. § 112, first paragraph, based on enablement is improper. Applicant further submits that the Section 112, first paragraph, rejection is improper because the Examiner did not individually address each independent claim alleged to be non-enabled. Moreover, Applicant submits that the features recited in claims 29-35 are properly enabled by Applicant's disclosure since several of the *In re Wands* factors are satisfied. Clearly, one skilled in the art at the time of Applicant's invention would be able to make and use Applicant's invention, as recited in claims 29-35, without undue experimentation.

For at least the foregoing reasons, Applicant respectfully requests that the rejection of pending claims 29-35 under 35 U.S.C. § 112, first paragraph, based on the enablement requirement, be reconsidered and withdrawn.

REJECTION UNDER SECTION 102(b) BASED ON REOHR

In paragraph 7 of the Office Action, the Examiner rejected claims 1, 3-5, 7-9, 11, 13, 15, 16, 18-22, and 24-35 under 35 U.S.C. § 102(b) as being clearly anticipated by the documentation of REOHR. Applicant respectfully traverses the rejection with regard to the claims presented

herein.

Applicant respectfully submits that the Section 102(b) rejection based on REOHR is improper because REOHR is not a "printed publication" under 35 U.S.C. § 102(b). A "publication" under 35 U.S.C. § 102(b) means "accessible to the interested public." *Carella v. Starlight Archery*, 804 F.2d 135, 231 U.S.P.Q. 644 (Fed. Cir. 1986) (holding that the characterizer of the information as a "printed publication" must provide proof of dissemination or the availability and accessibility of the item to the persons concerned with the art to which the document relates). A publicly displayed document where persons of ordinary skill in the art ***could see it*** and ***are not precluded from copying it*** can constitute a "printed publication," even if it is not disseminated by the distribution of reproductions or copies and/or indexed in a library or database. M.P.E.P. § 2128.01(IV) (emphasis added). Applicant submits that persons of ordinary skill in the art could not see and were precluding from copying REOHR, and thus, the reference is not a printed publication.

As set forth in the DECLARATION OF JOHN REOHR III UNDER 37 C.F.R. § 1.132 (hereinafter, "Declaration"), the reference referred to as REOHR was submitted to the Copyright Office by the inventor of the present application as a copyright deposit, and was believed to be received by the Copyright Office on January 3, 1997 (Declaration ¶¶ 3 and 4). Other than the copy submitted to the Copyright Office, no further copies of REOHR were disseminated by the inventor (Declaration ¶ 5).

Furthermore, Applicant notes that photocopies of copyright deposits (e.g., REOHR) could be provided by the Copyright Office only under 37 C.F.R. § 201.2(d)(2), which states:

> (2) Requests for certified or uncertified reproductions of the copies, phonorecords, or identifying material deposited in connection with a copyright registration of published or unpublished works in the custody of the Copyright Office will be granted only when one

> of the following three conditions has been met:
>
> (i) The Copyright Office receives written authorization from the copyright claimant of record or his or her designated agent, or from the owner of any of the exclusive rights in the copyright as long as this ownership can be demonstrated by written documentation of the transfer of ownership.
>
> (ii) The Copyright Office receives a written request from an attorney on behalf of either the plaintiff or defendant in connection with litigation, actual or prospective, involving the copyrighted work. The following information must be included in such a request:
>
> (A) The names of all the parties involved and the nature of the controversy;
>
> (B) The name of the court in which the actual case is pending or, in the case of a prospective proceeding, a full statement of the facts of the controversy in which the copyrighted work is involved; and
>
> (C) Satisfactory assurance that the requested reproduction will be used only in connection with the specified litigation.
>
> (iii) The Copyright Office receives a court order for reproduction of the deposited copies, phonorecords, or identifying material of a registered work which is the subject of litigation. The order must be issued by a court having jurisdiction of the case in which the reproduction is to be submitted as evidence.

According to this section of Code of Federal Regulations, photocopies of REOHR could be provided by the Copyright Office only upon specific authorization of the owner, for pending litigation, or by a court order. Applicant (the copyright owner or claimant) never provided authorization to anyone to order a photocopy of REOHR (Declaration ¶ 7). Applicant further submits that the copyright "A VIRTUALIZED NETWORK" was never the subject of a pending litigation, and as such, is not believed to have been the subject of a court order (Declaration ¶ 8).

In light of the above, Applicant submits that one of ordinary skill in the art of the present application could not see and was precluded from copying REOHR at all times (Declaration ¶ 9). Therefore, Applicant respectfully submits that REOHR is not a "printed publication" and cannot be used to form a proper Section 102(b) rejection of claims 29-35.

For at least the foregoing reasons, Applicant respectfully requests that the rejection of

pending claims 29-35 under 35 U.S.C. § 102(b) based on REOHR be reconsidered and withdrawn.

CONCLUSION

In view of the foregoing amendments and remarks, Applicant respectfully requests the Examiner's reconsideration of this application, and the timely allowance of the pending claims. Applicant respectfully requests entry of the present amendment because the present amendment does not raise new issues or require a further search of the art since the features were previously examined by the Examiner. Moreover, Applicant submits that the present amendment places the application in better condition for appeal should the Examiner contest the patentability of the pending claims.

If the Examiner does not believe that all pending claims are now in condition for allowance, the Examiner is urged to contact the undersigned to expedite prosecution of this application.

To the extent necessary, a petition for an extension of time under 37 C.F.R. § 1.136 is hereby made. Please charge any shortage in fees due in connection with the filing of this paper, including extension of time fees, to Deposit Account No. 50-1070 and please credit any excess fees to such deposit account.

Respectfully submitted,

HARRITY SNYDER, L.L.P.

By: /James M. Olsen/
James M. Olsen
Reg. No. 40,408

Date: September 25, 2006

11350 Random Hills Road
Suite 600
Fairfax, VA 22030
Phone: (302) 478-4548
Fax: (571) 432-0808